

# Analyse d'algorithmes d'optimisation

**OC Projet 7 :**

**Résolvez des problèmes en utilisant des algorithmes en Python**

Thème :

Algorithme pour optimiser les stratégies d'investissement pour les clients d'une société financière spécialisée dans l'investissement



Propulsé par  
M. EL-WALID EL-KHABOU



# TABLE DES MATIÈRES

## 1. Partie Algorithme

- Pseudo-code de l'algorithme dit de "Force Brute".
- Analyse de l'algorithme réel dit de "Force Brute".
- Processus de réflexion pour la solution optimisée.
- Pseudo-code de l'algorithme optimisé réel.
- Description et analyse de l'algorithme "optimisé" (avantages / limites).
- Comparaison de l'efficacité "Force Brute" VS " Optimisé".

## 2. Comparaison côte-à-côte avec les résultats de Sienna (Dataset 1 et 2).

## 3. Analyse empirique de la complexité des deux Algorithmes.



## Pseudo-code de l'algorithme dit de "Force Brute"

- 1. Définir MAX\_INVEST comme  $500 * 100$ . Cela représente le budget maximal que nous pouvons investir.**
- 2. Définir une fonction get\_csv\_data:**
  - Ouvrir le fichier brutforce.csv pour la lecture.
  - Pour chaque ligne du fichier (en ignorant la première ligne qui est l'en-tête):
    - Extraire le nom de l'action, le prix et le bénéfice.
    - Convertir le prix et le bénéfice de l'euro en centimes.
    - Renvoyer le nom de l'action, le prix en centimes et le bénéfice en centimes comme une ligne de données.
- 3. Définir une fonction generate\_combinations:**
  - Initialiser une variable profit à 0. Cela représente le profit maximal actuel.
  - Initialiser une liste best\_combination vide. Cela représente la meilleure combinaison d'actions actuelle.
  - Pour chaque sous-ensemble d'actions (l'algorithme de combinaisons, de taille 1 à la taille totale des actions):
    - Calculer le coût total de la combinaison d'actions.
    - Si le coût total est inférieur ou égal à MAX\_INVEST :
      - Calculer le profit total de la combinaison d'actions.
      - Si le profit total est supérieur au profit maximal actuel :
        - Mettre à jour le profit maximal et la meilleure combinaison d'actions.
- 4. Définir une fonction display\_result :**
  - Afficher le nom de chaque action, son prix et son bénéfice dans la meilleure combinaison d'actions.
  - Calculer et afficher la somme dépensée et le profit total.
- 5. Si le script est exécuté en tant que programme principal :**
  - Appeler la fonction get\_csv\_data pour obtenir les actions.
  - Appeler la fonction generate\_combinations pour obtenir la meilleure combinaison d'actions.
  - Appeler la fonction display\_result pour afficher le résultat.

## Analyse de l'algorithme dit de "Force Brute"

### Comment cet algorithme « brutforce » fonctionne t-il ?

- Il compose toutes les combinaisons possibles en fonction du nombre d'actions du Dataset
- Il calcul, pour chaque combinaison, son coût total et le compare au portefeuille du client (fixé à 500€) :
  - Si le coût est trop élevé, il supprime la combinaison
  - Si le coût est égal ou en dessous du montant du porte-feuilles du client, il l'ajoute dans une liste
- Il consulte la liste finale contenant les combinaisons valables et calcul, pour chacune de celles-ci, le bénéfice après 2 ans.
- Il ne conserve que le meilleur bénéfice et l'affiche en console.

### Les avantages de cette méthode :

- L'utilisateur a la garantie que l'algorithme va explorer toutes les combinaisons possibles
- L'utilisateur a la garantie de posséder, in fine, la combinaison la plus rémunératrice par rapport à son investissement

### Les Inconvénients de cette méthode :

- L'algorithme est lent (environ 12s de temps d'attente pour un dataset de 20 actions seulement)
- Suivant la taille du dataset, l'algorithme peut-être amené à crasher suite à un temps et un nombre de calcul trop long



## Processus de réflexion pour la solution optimisée

### De quoi est constituée une actions ?

- Une référence.
- Un prix à l'achat.
- Un bénéfice sur 2 ans exprimé en % en relation avec le prix d'achat  
=> prix d'achat de l'action  $X$  (bénéfice(%) / 100).

### Comment devrait se comporter l'algorithme de manière idéal ?

- Il devrait pouvoir donner la meilleure combinaison possible, peut-importe le nombre d'actions référencé.
- Il devrait pouvoir s'exécuter sans aucun crash.
- Il devrait pouvoir s'exécuter instantanément ( - de 1s dans le cahier des charges pour l'exemple).

# Pseudo-code de l'algorithme optimisé réel

## Début

- Initialiser my\_data\_files avec les chemins d'accès vers les fichiers de données
- Initialiser wallet avec le montant d'argent initial
- Initialiser first\_result comme un dictionnaire vide

## Définir la fonction read\_data(my\_data) :

- Lire le fichier CSV avec le chemin my\_data et créer un DataFrame df
- Créer une nouvelle colonne 'benefice' dans df par multiplication des colonnes 'price' et 'profit' divisée par 100
- Filtrer df pour ne garder que les lignes où 'price' est supérieur à 0, inférieur ou égal à wallet et 'benefice' supérieur à 0, trier ces lignes par 'profit' dans l'ordre décroissant et stocker le résultat dans df\_sorted
- Retourner df\_sorted

## Définir la fonction bag\_algorithm(df\_sorted, wallet) :

- Pour chaque ligne i dans df\_sorted :
- Récupérer le prix de l'action action à la ligne i
- Calculer test comme wallet - action
  - Si action est inférieur à wallet et test est supérieur à 0 :
  - Ajouter une nouvelle entrée au dictionnaire first\_result avec comme clé le nom de l'action et comme valeur une liste contenant le nom, le prix, le profit et le bénéfice de l'action
- Soustraire action à wallet

## Définir la fonction analyse(first\_result) :

- Convertir first\_result en DataFrame first\_result\_df avec 'name', 'price', 'profit' et 'benefice' comme colonnes
- Calculer first\_result\_sum\_price comme la somme des prix des actions dans first\_result\_df
- Calculer first\_result\_benefice comme la somme des bénéfices des actions dans first\_result\_df
- Créer actions comme une liste des noms des actions dans first\_result\_df
- Imprimer first\_result\_df
- Imprimer actions
- Imprimer first\_result\_sum\_price
- Imprimer first\_result\_benefice

## Définir la fonction main() :

- Pour chaque my\_data dans my\_data\_files :
- Effacer le contenu de first\_result2. Marquer le temps de début start\_time
- Appeler read\_data(my\_data) et stocker le résultat dans df\_sorted
- Appeler bag\_algorithm(df\_sorted, wallet)
- Appeler analyse(first\_result)
- Marquer le temps de fin end\_time
- Imprimer le temps d'exécution comme end\_time - start\_time

## Appeler main()

## Fin



## Algorithme : Description de l'algorithme “optimisé” (avantages / limites)

L'algorithme utilisé résout le problème dit du “rendu de monnaie”. La problématique de ce problème est d'arriver à rendre la monnaie sur un certain montant de la manière la plus efficace possible mais en limitant le nombre de pièces et billets à fournir.

Ici le montant représente le portefeuille du client, la monnaie à rendre va représenter le bénéfice sur 2 ans que cela peut lui apporter, et les pièces et billets vont quant à elles représenter les actions de nos dataset.

### Comment cet algorithme « Optimisé » fonctionne t-il ?

- Il va d'abord extraire toutes les actions une par une afin de calculer le bénéfice sur 2 ans maximum en euros (tout en triant les actions qui semblent incohérentes)
- Il va ensuite classer ses actions de la plus rentable à la moins rentable
- Il va ajouter les actions une par une (qui peuvent “rentrer”) dans le portefeuilles du client et additionner les bénéfices au fur et à mesure.

### Les avantages de cette méthode :

- L'utilisateur a la garantie que l'algorithme va inclure les actions à fort rendement
- L'algorithme sera beaucoup plus rapide (- de 1s de temps d'exécution)
- Le bénéfice final peut s'avérer être plus intéressant pour le client (voir comparaison avec Sienna)

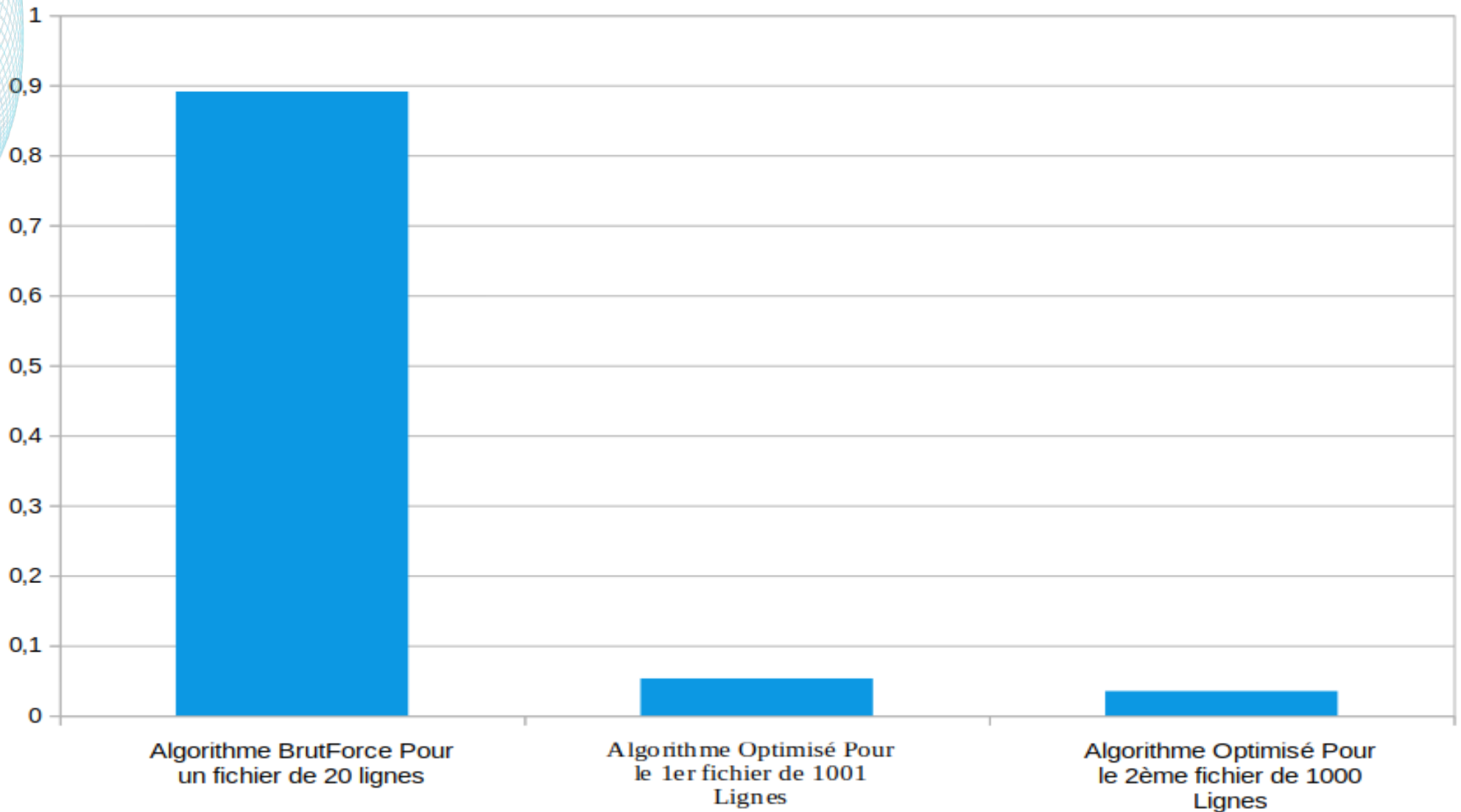
### Les Inconvénients de cette méthode :

- La combinaison proposée ne sera peut-être pas la plus optimale
- Suivant la taille du dataset, les résultats peuvent être assez différents d'une méthode à l'autre



## Algorithme : Comparaison de l'efficacité "Force Brute" VS "Optimisé"

Seconde(s)



Somme dépensée : 498.0€  
Profit total : 99.07€  
Temps d'Execution: 0.891 secondes

Total d'actions achetées : 499.94€  
Total des Bénéfices Net : 198.51€  
Temps d'exécution : 0.053 secondes

Total d'actions achetées : 499.98€  
Total des Bénéfices Net : 197.77€  
Temps d'exécution : 0.035 secondes



## Algorithme : Comparaison de l'efficacité "Force Brute" VS "Optimisé"

### Pourquoi observe-t-on une telle différence de rapidité ?

#### A) L'Algorithme « brutforce » :

L'algorithme de Force Brute utilise une méthode qui demande à l'ordinateur d'exécuter un nombre extrêmement élevé d'opérations et de combinaisons.

Dans le cas d'un ensemble de données de 20 actions, cet algorithme génère un total de 1 048 576 combinaisons, parmi lesquelles 813 347 coûtent moins de 500€ à l'achat, soit environ 77% de toutes les combinaisons.

De surcroît, l'ordinateur consacre une importante quantité de temps à la création des combinaisons, puis parcourt une seconde liste pour effectuer les calculs et comparaisons, ajoutant ainsi encore du temps supplémentaire.

Le temps nécessaire augmentera proportionnellement à la taille de l'ensemble de données, car la vitesse d'exécution est directement liée au nombre d'opérations à effectuer.

#### B) L'Algorithme « Optimisé » :

Contrairement à l'algorithme "Optimisé", ce dernier n'a pas besoin de générer de combinaisons. Il se contente de trier les actions de la plus rentable à la moins rentable en termes de bénéfices. Il examine ensuite chacune de ces actions une par une et compare son coût au portefeuille du client.

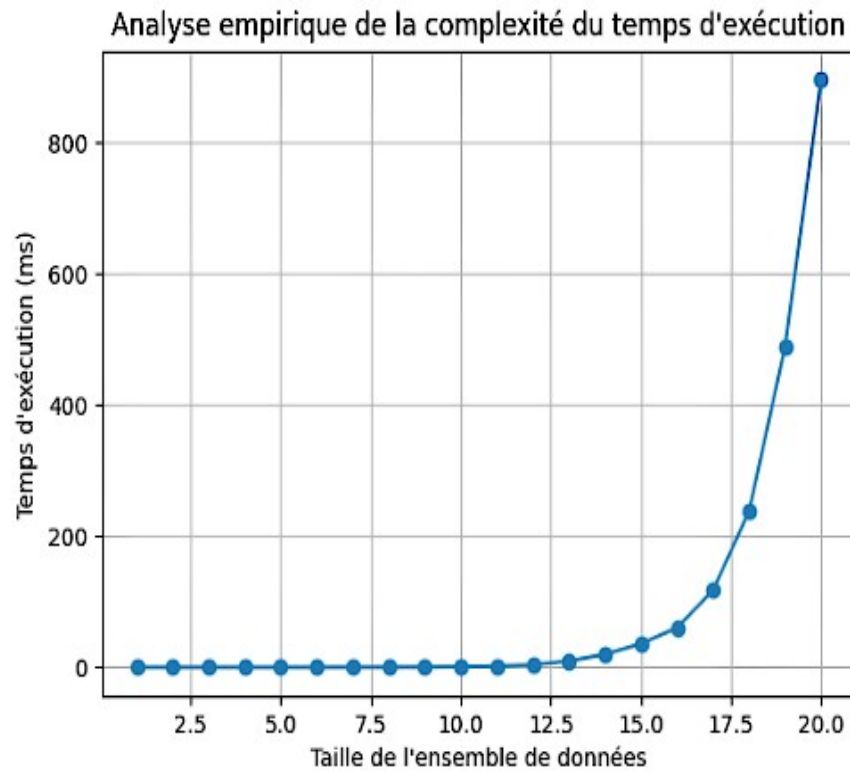
Si l'action est abordable, le programme l'ajoute à la liste finale. Il y a donc nettement moins d'opérations à effectuer avec cette méthode.

# Algorithme : Comparaison de l'efficacité "Force Brute" VS "Optimisé"

Comparaison des performances (méthode de notation Big-O)

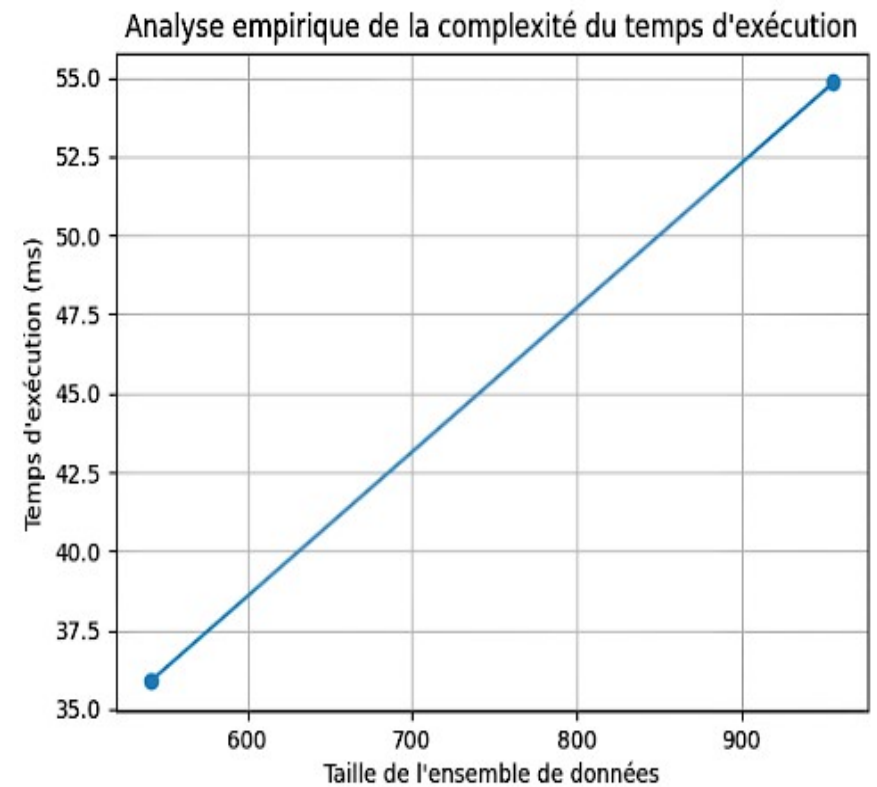
Algorithme de "Force Brute"

Notion BIG\_O :  $O(n^3)$



Algorithme "Optimisé"

Notion BIG\_O :  $O(n)$



## Comparaison côte-à-côte avec les résultats de Sienna (Dataset 1)

	name	price	profit	benefice
0	Share-XJMO	9.39	39.98	3.754122
1	Share-KMTG	23.21	39.97	9.277037
2	Share-MTLR	16.49	39.97	6.591053
3	Share-GTQK	15.40	39.95	6.152300
4	Share-LRBZ	32.90	39.95	13.143550
5	Share-WPLI	34.64	39.91	13.824824
6	Share-GIAJ	10.75	39.90	4.289250
7	Share-GHIZ	28.00	39.89	11.169200
8	Share-IFCP	29.23	39.88	11.656924
9	Share-ZSDE	15.11	39.88	6.025868
10	Share-FKJW	21.08	39.78	8.385624
11	Share-NHWA	29.18	39.77	11.604886
12	Share-LPDM	39.35	39.73	15.633755
13	Share-QQTU	33.19	39.60	13.143240
14	Share-USSR	25.62	39.56	10.135272
15	Share-EMOV	8.89	39.52	3.513328
16	Share-LGWG	31.41	39.50	12.406950
17	Share-QLMK	17.38	39.49	6.863362
18	Share-SKKC	24.87	39.49	9.821163
19	Share-UEZB	24.87	39.43	9.806241
20	Share-CBNY	1.22	39.31	0.479582
21	Share-CGJM	17.21	39.30	6.763530
22	Share-EVUW	4.44	39.22	1.741368
23	Share-FHZN	6.10	38.09	2.323490
24	Share-MLGM	0.01	18.86	0.001886
=====				
Total d'actions achetées :		499.94€		
Total des Bénéfices Net :		198.51€		
Temps d'exécution :		0.065 secondes		

Traitement du Dataset-1 avec notre  
Algorithme Optimisé

Sienna bought:

Share-GRUT

Total cost: 498.76â,-

Total return: 196.61â,-

Résultat de Sienna

Les résultats ne correspondent visiblement pas avec l'algorithme de Sienna.

Cependant nous pouvons ici remarquer que le bénéfice sur 2 ans est plus important avec notre algorithme optimisé (+ 1,90€) mais que le coût de cette combinaison est aussi plus élevée (+1,18€).

La différence coût supplémentaire VS Bénéfice supplémentaire reste néanmoins meilleure avec l'algorithme optimisé (+0,72€).

Le rendement de notre combinaison est de 39,70% contre 39,41% par rapport à l'algorithme de Sienna



## Comparaison côte-à-côte avec les résultats de Sienna (Dataset 2)

	name	price	profit	benefice
0	Share-PATS	27.70	39.97	11.071690
1	Share-ALIY	29.08	39.93	11.611644
2	Share-JWGF	48.69	39.93	19.441917
3	Share-NDKR	33.06	39.91	13.194246
4	Share-PLLK	19.94	39.91	7.958054
5	Share-FWBE	18.31	39.82	7.291042
6	Share-LFXB	14.83	39.79	5.900857
7	Share-ZOFA	25.32	39.78	10.072296
8	Share-ANFX	38.55	39.72	15.312060
9	Share-LXZU	4.24	39.54	1.676496
10	Share-FAPS	32.57	39.54	12.878178
11	Share-XQII	13.42	39.51	5.302242
12	Share-ECAQ	31.66	39.49	12.502534
13	Share-JGTW	35.29	39.43	13.914847
14	Share-IXCI	26.32	39.40	10.370080
15	Share-DWSK	29.49	39.35	11.604315
16	Share-ROOM	15.06	39.23	5.908038
17	Share-VCXT	29.19	39.22	11.448318
18	Share-YFVZ	22.55	39.10	8.817050
19	Share-OCKK	3.16	36.39	1.149924
20	Share-JMLZ	1.27	24.71	0.313817
21	Share-DYVD	0.28	10.25	0.028700

=====

Total d'actions achetées : 499.98€  
 Total des Bénéfices Net : 197.77€  
 Temps d'exécution : 0.036 secondes



Traitement du Dataset-2 avec notre  
Algorithme Optimisé



Résultat de Sienna

Sienna bought:

Share-ECAQ 3166  
 Share-IXCI 2632  
 Share-FWBE 1830  
 Share-ZOFA 2532  
 Share-PLLK 1994  
 Share-YFVZ 2255  
 Share-ANFX 3854  
 Share-PATS 2770  
 Share-NDKR 3306  
 Share-ALIY 2908  
 Share-JWGF 4869  
 Share-JGTW 3529  
 Share-FAPS 3257  
 Share-VCAX 2742  
 Share-LFXB 1483  
 Share-DWSK 2949  
 Share-XQII 1342  
 Share-ROOM 1506

Ici les résultat semblent plus proche entre les deux algorithmes (22 actions contre 18 pour Sienna) puisque plusieurs actions se retrouvent dans les deux combinaisons (ECAQ - IXCI - FWBE - NDKR ...)

Cependant nous pouvons ici remarquer que le bénéfice sur 2 ans est plus important avec notre algorithme optimisé (+ 4,00 €) mais que le coût de cette combinaison est aussi plus élevée (+10,74€).

La différence coût supplémentaire VS Bénéfice supplémentaire est clairement déficitaire (-6,74€ par rapport à la combinaison de Sienna).

Malgré tout nous constatons que le rendement de notre algorithme est de 39,55% contre 39,60% pour celui de Sienna ce qui ne représente pas une différence flagrante

Total cost: 489.24â,-  
 Profit: 193.78â,-